Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 387 (2007) 348–359
Theoretical Computer Science
www.elsevier.com/locate/tcs

Optimal lower bounds for rank and select indexes

Alexander Golynski

School of Computer Science, University of Waterloo, 200 University Avenue West, Waterloo, ON, N2L3G1, Canada

Abstract

We develop a new lower bound technique for data structures. We show an optimal $\Omega(n \lg \lg n / \lg n)$ space lower bounds for storing an index that allows to implement rank and select queries on a bit vector B provided that B is stored explicitly. These results improve upon [Peter Bro Miltersen, Lower bounds on the size of selection and rank indexes, in: Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms, 2005, pp. 11–12]. We show $\Omega((m/t) \lg t)$ lower bounds for storing rank/select index in the case where B has m 1-bits in it and the algorithm is allowed to probe t bits of B. We also present an improved data structure that implements both rank and select queries with an index of size $(1 + o(1))(n \lg \lg n / \lg n) + O(n / \lg n)$, that is, compared to existing results we give an explicit constant for storage in the RAM model with word size $\lg n$. An advantage of this data structure is that both rank and select indexes share the most space consuming part of order $\Theta(n \lg \lg n / \lg n)$ making it more practical for implementation.

Keywords: Fully indexable dictionary; Lower bounds; Succinct data structures

1. Introduction

The term *succinct data structure* was first used by Jacobson in [2], where he defined and proposed a solution to the following problem of implementing rank and select queries. We are given a bit vector B of length n. The goal is to represent B in such a way that rank and select queries about B can be answered efficiently. Query $\text{rank}_B(i)$ returns the number of 1-bits in B before (and including) the position i, and $\text{select}_B(i)$ query returns the position of the ith occurrence of 1 in B. We require that the representation should be succinct, that is, the amount of space S it occupies is close to the information-theoretic minimum, namely $S = n + o(n)$ in the case of bit vectors of length n. We consider this problem in the RAM model with word size $w = \Theta(\lg n)$. Jacobson proposed a data structure to perform rank queries that uses $n + O(n \lg \lg n / \lg n)$ bits of space and requires only $O(1)$ time to compute the answer. His implementation of the select query requires $O(\lg n)$ bit accesses, but it does not take advantage of word parallelism and runs in time that is more than a constant in the RAM model. It was subsequently improved by Clark [1], Munro et al. [4,5], and Raman et al. [6]. The index proposed by Raman et al. [6] occupies $O(n \lg \lg n / \lg n)$ bits, and the select query is implemented in $O(1)$ time.

An indexing data structure stores data in "raw form" (i.e. B is stored explicitly) plus a small index I to facilitate implementation of queries, such as rank and select. We denote the size of the index by r.

E-mail address: agolynski@cs.uwaterloo.ca.

0304-3975/$ - see front matter
doi:10.1016/j.tcs.2007.07.041

Miltersen [3] showed that any indexing data structure that allows $O(1)$ time implementation of rank (select) queries must use an index of size at least $\Omega(n \lg\lg n/\lg n)$ bits (respectively $\Omega(n/\lg n)$ bits). The purpose of this paper is to develop a new technique for showing lower bounds for indexing data structures. This technique allows to improve the lower bounds of Miltersen [3] for both rank and select problems to match the corresponding upper bounds.

For our lower bounds, we use the quite standard *indexing model* which can be described as follows. Our goal is to store and perform a set of queries on a given family of combinatorial objects (e.g. rank/select on bit vectors). Let B be a "raw" representation of a combinatorial object (e.g. a raw bit vector). We assume that B is given to us free of charge, e.g. it is stored in external memory such as a large database; or is provided by outside world, e.g. web graph [7]. Let I be an index that helps in performing the set of queries efficiently; presumably it is stored in a relatively fast, expensive and/or limited memory. We are charged 1 unit of space for each bit in I, while access to I is free of charge. An algorithm that performs a query has unlimited computation power, however we are charged 1 unit of time when it accesses (e.g. probes one bit in) B.

We show that any algorithm in the indexing model that performs rank (respectively select) queries with the time cost $O(\lg n)$, must have the space cost at least $\Omega(n \lg\lg n/\lg n)$. Note that this setting is general enough; in particular, it subsumes $O(1)$ time RAM algorithms with word size $O(\lg n)$. Hence for the select index, we improve the lower bound of Miltersen to the optimal; and for the rank index, we show the same bound, but in a more general setting.

We also consider the case where the number of 1-bits in a bit vector B is fixed, we denote it by m and call it *cardinality*. In this setting, for both rank and select problems, we prove that any algorithm with the time cost t has the space cost $\Omega((m/t)\lg t)$. In particular, this lower bound is optimal for bit vectors of constant 0th order entropy, i.e. $m = \Theta(n)$; and it yields strong lower bounds in the case where $m = \Omega(n/\lg\lg n)$.

We also give an index that allows to implement both rank and select queries in $O(1)$ time and uses space $(1 + o(1))(n \lg\lg n/\lg n) + O(n/\lg n)$. Thus, we give an explicit constant in front of the leading term $n \lg\lg n/\lg n$ compared to the previous best result of Raman et al. [6] that gives an index of size $O(n \lg\lg n/\lg n)$. This index is simple and space efficient, and it might be of interest to practitioners.

This paper is organized as follows. In Section 2, we give an implementation for rank and select queries. In Section 3 we prove lower bounds for rank and select indexes. In Section 4 we generalize the lower bounds from Section 3 to the case of bit vectors with a given cardinality.

2. Upper bounds

In this section, we will improve the result of Raman et al. [6] that gives an optimal index for the select query of size $O(n \lg\lg n/\lg n)$. We assume that the word size is $w = \lg n$ for the part of the paper that concerns with upper bounds; in contrast, all lower bounds are shown in the indexing model (bit probes).

Then we will construct an optimal index of size $(1 + o(1))(n \lg\lg n/\lg n) + O(n/\lg n)$ for rank especially query. A similar result was obtained by Jacobson [2]; however we implement both the rank and the select indexes simultaneously, such that the space used is just $n + (1 + o(1))(n \lg\lg n/\lg n) + O(n/\lg n)$.

Both of these indexes share a component of size $(1 + o(1))(n \lg\lg n/\lg n)$ that we call a *count index*. The count index is constructed as follows: we split our bit vector B into *chunks* of size $\lg n - 3\lg\lg n$. Then we store the number of 1-bits in each chunk (we call it *cardinality of a chunk*) in equally spaced fields of size $\lg\lg n$ for a total of $n \lg\lg n/(\lg n - 3\lg\lg n) = (1 + o(1))n \lg\lg n/\lg n$ bits.

2.1. Optimal select index

In this subsection, we describe a new select index that uses the count index plus $O(n/\lg n)$ additional bits. Let B be the bit vector of length n. Let $S_1 = (\lg n)^2$. We store the locations of each (iS_1)th occurrence of 1-bit in B, for each $1 \le i \le n/S_1$. Note that normally the number of 1-bits in B is less than n, so part of the range is unused. This takes $O(n/\lg n)$ bits in total. We call regions from position $\text{select}(iS_1)$ to position $\text{select}((i+1)S_1) - 1$ *upper blocks*. To perform $\text{select}_B(i)$, we first compute $j = \lfloor i/S_1 \rfloor$ the number of the upper block that the ith bit is in, so that

$$\text{select}_B(i) = \text{select}_B(jS_1) + \text{select}_{UB_j}(i \bmod S_1),$$

where select_{UB_j} denotes the select query with respect to the jth upper block. We call such an operation *reduction from cardinality n to S_1*. Now we need to implement the select query for upper blocks. We call an upper block *sparse* if its

length is at least $(\lg n)^4$. For a sparse block, we can just explicitly write the answers for all the possible select queries, this will use at most $(\lg n)^3$ bits. Intuitively, this is roughly at most $O(1/\lg n)$ indexing bits per one bit from B, so that the total space used up by this part of the index sums up to $O(n/\lg n)$ bits. We will repeatedly use this $1/\lg n$ rule.

Let us consider a non-sparse upper block. It is a bit vector of cardinality S_1 and length at most $(\lg n)^4$. Thus, it takes $O(\lg\lg n)$ bits to encode a pointer within such a block. We perform cardinality reduction from S_1 to $S_2 = \lg n \lg\lg n$. Similar to upper blocks, we introduce *middle blocks*, each having cardinality S_2. That is, encode every (iS_2)th occurrence of 1-bit in an upper block. This information occupies $O(\lg\lg n \cdot \lg n/\lg\lg n) = O(\lg n)$ bits for an upper block of length at least $(\lg n)^2$, so that we use $1/\lg n$ bits for index per one bit from B, for a total of at most $O(n/\lg n)$ bits. We call a middle block *sparse* if it has length more than $(\lg n \lg\lg n)^2$. If a middle block is sparse, then we can explicitly write the positions of all the occurrences of 1-bits in it, this uses at most $\lg n(\lg\lg n)^2$ bits (at most $O(1/\lg n)$ indexing bits per one original bit). We call a middle block *dense* if its length is at most $\frac{(\lg n)^2}{4\lg\lg n}$.

If a middle block is neither sparse nor dense, then use cardinality reduction from S_2 to $S_3 = (\lg\lg n)^3$. Call the resulting blocks of cardinality S_3 *lower blocks*. That is, store every (iS_3)th occurrence of 1 in a middle block. This uses $\lg n/\lg\lg n$ bits per block of length at least $\frac{(\lg n)^2}{2\lg\lg n}$, hence $O(1/\lg n)$ indexing bits per original bit. We say that a lower block is *sparse* if it has length at least $\lg n(\lg\lg n)^4$, and *dense* otherwise. If a lower block is sparse, then we can explicitly encode all 1-bit occurrences in it.

It remains to implement select query for dense middle and lower blocks. Consider, for example, a dense middle block MB and implement $\text{select}_{MB}(i)$ on it. We first assume that MB is aligned with chunks, i.e. its starting (ending) position coincide with starting (ending) position of some chunk (chunks are of size $\lg n - 3\lg\lg n$). Recall that the length of MB is at most $(\lg n)^2/4\lg\lg n$, so that the part P of the count index that covers MB (i.e. P encodes cardinality of each chunk inside MB) is of size at most $(\lg n)/2$. Hence, we can read P in one word and perform a lookup in a table T to compute the number of the chunk where the ith 1-bit of MB is located. Table T is of size at most $\sqrt{n}\lg n \lg\lg n \times \lg\lg n$, and it stores for each possible choice of P and for each $j = O(\lg n \lg\lg n)$ the number of the chunk where the jth occurrence of 1 is located (denote the corresponding chunk by C), and the rank of that occurrence inside C (denote it by p). Now we can compute $\text{select}_{MB}(j)$ by reading chunk C and performing $\text{select}_C(p)$ using a lookup in a table Q. Table Q is of size at most $O(2^{\lg n - 3\lg\lg n} \cdot \lg n \times \lg\lg n) = O(n/\lg n)$, and it stores for each possible chunk C and for each position k the result of $\text{select}_C(k)$. The case where MB is not aligned with chunks can be resolved by counting number of 1-bits in the first chunk that partially belongs to MB (e.g. a lookup in a table that computes rank within a chunk, we discuss this table later in the next subsection) and adjusting j accordingly. Clearly, select query for the case of dense lower blocks can be implemented in the same way.

2.2. Optimal rank index

In this subsection, we show how to design the rank index using the count index and $O(n/\lg n)$ additional bits.

We divide the bit vector B into equally sized *upper blocks* of size $S_1 = (\lg n)^2$ bits each. For each upper block, we write the rank of the position preceding its first position ($\text{rank}_B(0) = 0$). This information uses $O(n/\lg n)$ bits in total. Now we can compute $\text{rank}_B(i)$ as follows: first we compute $j = \lfloor i/S_1 \rfloor$, the number of the upper block that contains the ith bit of B (denote the upper chunk by UC), so that

$$\text{rank}_B(i) = \text{rank}_B(jS_1 - 1) + \text{rank}_{UC}(i \bmod S_1).$$

We call such an operation a *length reduction* from n to S_1. Then we perform another length reduction from S_1 to $S_2 = \lg n \lg\lg n$. We call the corresponding blocks of length S_2 *middle blocks*. It takes $\lg n$ bits for each upper block of length $(\lg n)^2$ to describe the ranks of the starting positions of middle blocks (each rank uses $\lg\lg n$ bits), so that we use $1/\lg n$ bits for index per one bit of B. Without loss of generality, we can assume that middle blocks are always aligned with chunks. Let MB be a middle block, we implement $\text{rank}_{MB}(i)$ as follows. Let $j = O(\lg\lg n)$ be the number of the chunk (denote it by C) that contains the ith bit of MB, $j = \lfloor i/S_3 \rfloor$, where $S_3 = \lg n - 3\lg\lg n$ denotes the length of a chunk. One middle block of size S_2 corresponds to a part P of counting index of size at most $O((\lg\lg n)^2)$ bits, so that we can read it in one word and use one lookup in a table T to compute $\text{rank}_{MB}(jS_3 - 1)$. Table T is of size $(\lg n)^{O(1)} \lg\lg\lg n \times \lg\lg n$, and it stores $\text{rank}_{MB}(jS_3 - 1)$ for each possible part P and chunk number j. Thus,

$$\text{rank}_{MB}(i) = \text{rank}_{MB}(jS_3 - 1) + \text{rank}_C(j \bmod S_3)$$

and the latter rank can be also computed by one lookup in the table Q. Recall that table Q of size $O(n/\lg n)$ stores for each possible chunk C and for each position k the result of $\text{select}_C(k)$.

3. Lower bounds

In this section, we consider lower bounds for rank and select algorithms in the indexing model with the time cost $O(\lg n)$, we denote the space cost (i.e. index size) by r.

3.1. Rank index

In this subsection, we develop a new combinatorial technique for proving lower bounds; and use it to show that the space cost for rank index is at least

$$r = \Omega(n \lg\lg n / \lg n) \tag{1}$$

if the time cost is bounded by $O(\lg n)$.

Let us fix the mapping between bit vectors B and indexes I and fix an algorithm that performs the rank query (i.e. it computes $\text{rank}_B(p)$ for a given p). As we mentioned before, an algorithm is allowed to perform an unlimited number of bit probes to I and has unlimited computation power; we only limit the number of bit probes it can perform to the bit vector B. Let us fix the number of bit probes $t = f \lg n$ for some constant $f > 0$. We split the bit vector B into p *blocks* of size $k = t + \lg n$ each. Let n_i be the number of 1-bits in the ith block, we call n_i the *cardinality* of ith block. For each block i, $1 \le i \le p$, we simulate the rank query on the last position of the ith block, $s_i = \text{rank}_B(ik - 1)$, so that $n_i = s_{i+1} - s_i$. Note that we will have at least $n - pt = p \lg n = \Omega(n)$ unprobed bits after the computation is complete. Now we will construct a binary *choices tree*. The first r levels correspond to all possible choices of index. At each node at depth r of the tree constructed so far, we will attach the decision tree of the computation that rank algorithm performed for the query $\text{rank}_B(k-1)$ when the index I is fixed. The nodes are labeled by the positions in the bit vector that the algorithm probes and the two outgoing edges are labeled 0 or 1 depending on the outcome of the probe; we call the corresponding probe a 0-*probe* or 1-*probe* respectively. At each leaf of the previously constructed tree, we attach the decision tree for $\text{rank}_B(2k-1)$ and so on. Thus, the height of the tree is at most $r + tp$. If the computation probes the same bit b twice (even if the previous probe was performed for a different rank query), we do not create a binary node for the second and latter probes; instead we use the result of the first probe to b. By definition of the tree, at the leaves, all block cardinalities n_i are known. Let us fix a leaf x, we call a bit vector B *compatible* with x iff: (1) the index (i.e. the first r nodes) on the root to the leaf path corresponds to the index for B; and (2) the remaining nodes on the root to the leaf path correspond to the choices made by the computation described above.

Let us bound the number $C(x)$ of bit vectors B that are compatible with a given leaf x (in what follows we will use C to denote $C(x)$).

Let u_i be the number of unprobed bits in the block i, so that $u_i \le k$ and

$$\sum_{i=1}^{p} u_i = U,$$

where U is the total number of unprobed bits. At a given leaf, we have computed all n_i's, and hence the sum of all unprobed bits (denote it by v_i) in the block i equals to n_i minus the number of 1-probes in the ith block. Therefore, we can bound the number of bit vectors compatible with x by

$$\frac{C}{2^U} \le \frac{\binom{u_1}{v_1}}{2^{u_1}} \frac{\binom{u_2}{v_2}}{2^{u_2}} \cdots \frac{\binom{u_p}{v_p}}{2^{u_p}}. \tag{2}$$

Let us classify blocks into two categories: determined and undetermined. We call the block i *determined* if $u_i \le U/(2p)$ (intuitively, when it has less than half of the "average" number of unprobed bits) and call it *undetermined* otherwise. Let d be the number of determined blocks. Then

$$U = \sum_i u_i \le dU/(2p) + (p-d)k$$

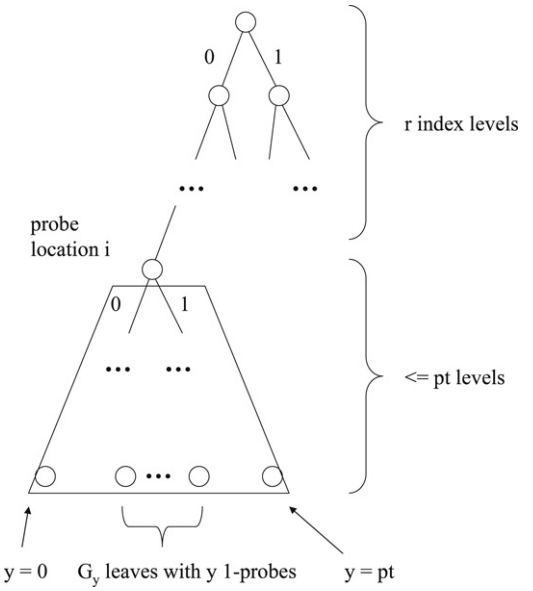

Fig. 1. Choices tree. The leaves could be at different levels. Notation G_y will be defined and used later in Section 4.

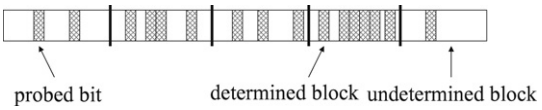

Fig. 2. Bit vector at a leaf.

hence

$$d \le p\frac{k - U/p}{k - U/(2p)} \le (1-a)p,$$

where $0 < a < 1$ is a constant. Thus, there is at least a constant fraction of undetermined blocks. We bound $\binom{u_i}{v_i}/2^{u_i} < 1$ for determined blocks, and

$$\frac{\binom{u_i}{v_i}}{2^{u_i}} \le \frac{\binom{u_i}{u_i/2}}{2^{u_i}} < \frac{b}{\sqrt{u_i}} \le \frac{b}{\sqrt{U/(2p)}} < \frac{c}{\sqrt{\lg n}}$$

for undetermined blocks using Stirling formula, where $b > 0$ and $c > 0$ are constants. Thus (2) can be bounded by

$$\frac{C}{2^U} \le \left(\frac{c}{\sqrt{\lg n}}\right)^{ap}. \tag{3}$$

Recall that both C and U depend on x, so that $U(x) = n + r - \mathrm{depth}(x)$. We can compute the following sum

$$\sum_{x \text{ is a leaf}} 2^{U(x)} = 2^{n+r} \sum_{x \text{ is a leaf}} 2^{-\mathrm{depth}(x)} = 2^{n+r}.$$

The total number of bit vectors B compatible with some leaf is at most

$$\sum_{x \text{ is a leaf}} C(x) \le 2^{n+r}\left(\frac{c}{\sqrt{\lg n}}\right)^{ap}. \tag{4}$$

However, each bit vector has to be compatible with at least one leaf

$$2^n \le \sum_{x \text{ is a leaf}} C(x).$$

Thus

$$r = \Omega(n \lg\lg n / \lg n).$$

The index presented in the previous section matches this lower bound up to a constant factor.

Note that the techniques given by Miltersen [3] do not allow to obtain the bound (1) in the case where we can perform $O(\lg n)$ bit probes to the bit vector B; although in a more restricted case where only $O(1)$ word probes are allowed his lower bound (1) is optimal. Miltersen [3] showed that the rank index has to be of size r, such that

$$2(2r + \lg(w+1))tw \geq n \lg(w+1),$$

where w denotes the word size, t denotes the number of word probes, and r is the size of an index. Miltersen reduces (i) a set of $\Omega(n/\lg n)$ independent problems; problem i is to compute $n_i \bmod w$, where n_i the number of 1-bits in the region $[2ciw, 2c(i+1)w]$, where c is a constant to
(ii) the problem of computing rank for positions $2ci \lg n$ for all i, $0 \leq i \leq n/(2c \lg n)$.

In each region of (i), $2c$ numbers $j_1, j_2, \ldots, j_{2c}$, such that $0 \leq j_k \leq w$ are encoded using unary representation $1^{j_k}0^{w-j_k}$. He shows a lower bound (1) for (i) when $w = \Theta(\lg n)$. For the case $w = 2$, this method only yields $r = \Omega(n/\lg n)$. One can try to generalize Miltersen's approach to allow $O(\lg n)$ bit probes instead of $O(1)$ word probes. The difficulty is that in the bit probe model, a number that is less than $\lg n$ when represented in unary can be recognized using binary search in $\lg\lg n$ bit probes, so that each independent problem of (i) can be solved in $O(\lg\lg n)$ bit probes without using an index. One can also try to "shuffle" bits in the unary representation to disallow such binary searches, however it is not clear whether such a proof can be completed. One could try an approach, where regions of (i) are of the length $\Omega(\lg n)$ and $w = 2$, however in such a case it suffices to store all the $O(n/\lg n)$ answer bits as the index, so that we do not need to perform any bit probes.

3.2. Select index

In this subsection, we apply a similar combinatorial technique to show an optimal lower bound for the select index. Fix the number of probes to the bit vector B to be $t = f \lg n$ (for some constant $f > 0$) that select algorithm uses and let $k = t + \lg n$ as before.

Let us restrict ourselves to bit vectors B of cardinality $n/2$ ($n/2$ bits are 0 and $n/2$ bits are 1). Let us perform the following $p = n/(2k)$ queries: for each $1 \leq i \leq p$ we simulate select(ik). Similarly, we construct choices tree for these queries. To compute the number of compatible bit vectors for a given leaf, we split each vector B into p *blocks*, where ith block starts at position $\text{select}_B((i-1)k) + 1$, and ends at position $\text{select}_B(ik)$ (we define $\text{select}_B(0) = 0$ for convenience). Note that there are exactly k ones in each block. The total number of unprobed bits U is at least $n - pt = n(1 - t/k) = \Omega(n)$. We can count the number of compatible nodes C for each leaf x by applying the same technique as for rank, and obtain (similar to (3))

$$\frac{C}{2^U} \leq \left(\frac{c}{\sqrt{\lg n}}\right)^{ap},$$

where $0 < a < 1$ and $0 < c$ are positive constants. Next, we can obtain the bound on the total number of bit vectors B that are compatible with at least one node in the choices tree. Similar to (4), we have

$$2^{n+r}\left(\frac{c}{\sqrt{\lg n}}\right)^{ap}.$$

The total number of bit vectors we are considering is $\binom{n}{n/2}$, thus

$$\binom{n}{n/2} \leq 2^{n+r}\left(\frac{c}{\sqrt{\lg n}}\right)^{ap}$$

and hence

$$r = \frac{\lg\binom{n}{n/2}}{n}\Omega\left(\frac{n\lg\lg n}{\lg n}\right) = \Omega\left(\frac{n\lg\lg n}{\lg n}\right).$$

Now we give an argument that techniques from Miltersen [3] cannot be improved from $r = \Omega(n/\lg n)$ to the optimal $r = \Omega(n \lg\lg n/\lg n)$. Miltersen used only bit vectors that only have $O(n/\lg n)$ 1-bits. However, for such vectors, we can construct an index of size $O(n/\lg n)$ that allows $O(1)$ select queries. Let us divide B into p subregions of size $(\lg n)/2$, for each subregion we count the number of 1 bits in it (denote it by n_i) and represent it in unary. We construct the following bit vector

$$L = 1^{n_1}01^{n_2}0\ldots01^{n_p}$$

of length $p + O(n/\lg n) = O(n/\lg n)$. To perform $\text{select}_B(j)$ on B, we first find $x = \text{select}_L(i)$ and then $i = x - \text{rank}_L(x)$, the number of 0-bits before the position x in L. Hence i gives us the number of blocks of B where jth 1-bit is located (denote the block K). Next, we compute $z = \text{select0}_L(i)$ (where $\text{select0}_L(j)$ gives position of jth occurrence of 0-bit in L), the starting position of ith block in L. And then compute $t = x - z$, so that jth 1-bit of B is tth 1-bit of K. Finally, $\text{select}_K(j)$ can be done by a lookup in a table of size $\sqrt{n}(\lg\lg n)^2$ bits that stores results of all possible select queries for all possible blocks. Note that rank and select on L requires at most $o(n/\lg n)$ bits in addition to storing L as we discussed in the previous section. Thus, the total space requirement for the index is $O(n/\lg n)$ bits. It follows that for such bit vectors B select indexes of size $O(n/\lg n)$ are optimal.

We state the results for the rank and the select indexes as the following

Theorem 3.1. *Let B be a bit vector of length n. Assume that there is an algorithm that uses $O(\lg n)$ bit probes to B plus unlimited access to an index of size r and unlimited computation power to answer rank (respectively, select) queries. Then $r = \Omega(\frac{n \lg\lg n}{\lg n})$.*

4. Density-sensitive lower bounds

In this section, we consider the case where the bit vector B contains some fixed number m of 1-bits and express lower bounds for the rank and select indexes in terms of both parameters m and n, the length of B. We will use techniques similar to the previous section; however, the calculations are slightly more involved in this case. We will prove a lower bound for the rank index first. Throughout this section, we will use the same notation as in Section 3: p denotes the number of simulations of rank (select) queries; k denotes the size of a block; t denotes the running time of a rank query; $U(x)$ (and its short form U) denotes the number of unprobed bits for a given leaf x of the choices tree; r denotes the size of the rank index.

First, assume that all leaves in the choices tree are at the same level $pt + r$, i.e. on every root to leaf path the rank algorithm probes exactly pt bits. If some node x is z levels above it, we can perform z fake probes, in order to split it into 2^z nodes at the required level. Thus, $U = n - pt$ for all leaves. Let us choose p, such that $pt \le n/2$, so that at least half of the bits are unprobed at every leaf. We will partition all the leaves x into m groups depending on the total number of 1-probes performed on the bit vector on the root to leaf path to x (that is, excluding the first r levels occupied by the index). Denote G_y the group of leaves for which we performed exactly y 1-probes. Clearly, $|G_y| \le 2^r\binom{pt}{y}$. For each leaf $x \in G_y$ we can bound the number of compatible bit vectors by:

$$\binom{u_1}{v_1}\binom{u_2}{v_2}\cdots\binom{u_p}{v_p}, \tag{5}$$

where

$$u_1 + u_2 + \cdots + u_p = U \tag{6}$$

$$v_1 + v_2 + \cdots + v_p = V, \tag{7}$$

where $U = n - pt$ is the number of unprobed bits and $V = m - y$. Similar to Section 3, let u_i denote the number of unprobed bits in ith block and $v_i \le u_i$ denote the sum of 1-bits in it. Recall that v_i equals to n_i minus number of 1-probes in ith block, and hence is fixed for a given leaf, i.e. the same for all the bit vectors compatible with the leaf.

We will combine blocks into larger *superblocks* as follows. The 1st superblock will contain blocks $1, 2, \ldots, z_1$, such that $k \le u_1 + u_2 + \cdots u_{z_1} \le 2k$, the ith superblock (except, perhaps, the last one that could be smaller than k) will contain blocks $z_{i-1} + 1, \ldots, z_i$ such that $k \le u_i^s \le 2k$, where

$$u_i^s = u_{z_{i-1}+1} + u_{z_{i-1}+2} + \cdots + u_{z_i}.$$

Note that this is always possible, since $u_i \le k$ for all i. Let q be the number of superblocks, then $n/(4k) \le U/(2k) \le q \le U/k \le n/k$, since $U \ge n/2$; or, equivalently, $p/4 \le q \le p$.

For each superblock, we will use the inequality

$$\binom{u_{z_{i-1}+1}}{v_{z_{i-1}+1}}\binom{u_{z_{i-1}+2}}{v_{z_{i-1}+2}}\cdots\binom{u_{z_i}}{v_{z_i}} \le \binom{u_i^s}{v_i^s},$$

where $v_i^s = v_{z_{i-1}+1} + v_{z_{i-1}+2} + \cdots + v_{z_i}$. So that

$$\binom{u_1}{v_1}\binom{u_2}{v_2}\cdots\binom{u_p}{v_p} \le \binom{u_1^s}{v_1^s}\binom{u_2^s}{v_2^s}\cdots\binom{u_q^s}{v_q^s}. \tag{8}$$

To bound this expression, we will maximize it over all possible v_i^s's with u_i^s's fixed, subject to the constraint

$$v_1^s + v_2^s + \cdots + v_q^s = V.$$

The point $(v_1^s, v_2^s, \ldots, v_q^s)$ is a *local maximum* if we cannot increase some v_i^s by 1 and decrease some other v_j^s by 1, so that (8) increases. The following simple lemma characterizes the local maxima:

Lemma 4.1. *At a local maximum, we have*

$$\frac{v_j^s + 1}{u_j^s + 1} \ge \frac{v_i^s}{u_i^s + 1}$$

for each $i \ne j$.

Proof.

$$\binom{u_i^s}{v_i^s}\binom{u_j^s}{v_j^s} \ge \binom{u_i^s}{v_i^s - 1}\binom{u_j^s}{v_j^s + 1} \quad \text{simplifies to}$$

$$\frac{u_i^s - v_i^s + 1}{v_i^s} \ge \frac{u_j^s - v_j^s}{v_j^s + 1} \quad \text{and the result follows.} \quad \square$$

Corollary 4.1. *At a local maximum, we have*

$$\left|\frac{v_i^s}{u_i^s} - \frac{V}{U}\right| \le \frac{2}{k}.$$

Proof. First note that for each $i \ne j$, we have

$$\frac{v_j^s}{u_j^s + 1} + \frac{1}{u_j^s + 1} \ge \frac{v_i^s}{u_i^s + 1} \ge \frac{v_j^s}{u_j^s + 1} - \frac{1}{u_i^s + 1}.$$

Since u_i^s and u_j^s are at least k, we have

$$\left|\frac{v_i^s}{u_i^s + 1} - \frac{v_j^s}{u_j^s + 1}\right| < \frac{1}{k}.$$

Since v_i^s/u_i^s and v_j^s/u_j^s are at most 1, it follows that

$$\left|\left(\frac{v_i^s}{u_i^s + 1} - \frac{v_j^s}{u_j^s + 1}\right) - \left(\frac{v_i^s}{u_i^s} - \frac{v_j^s}{u_j^s}\right)\right| = \left|\frac{v_i^s}{u_i^s(u_i^s + 1)} - \frac{v_j^s}{u_j^s(u_j^s + 1)}\right| < \frac{1}{k}.$$

Therefore,

$$\left|\frac{v_i^s}{u_i^s} - \frac{v_j^s}{u_j^s}\right| < \frac{2}{k}.$$

Finally, we observe that

$$\min\left\{\frac{v_1^s}{u_1^s},\ldots,\frac{v_q^s}{u_q^s}\right\}\le\frac{V}{U}\le\max\left\{\frac{v_1^s}{u_1^s},\ldots,\frac{v_q^s}{u_q^s}\right\}$$

and the result follows. □

To bound the expression (8) further, let us assume that the density of the bit vector is bounded by a constant $m/n\le d$, where $0<d<1$. Also assume that $t=\omega(1)$. We choose the parameter $p=m(1-d)/2t$. It follows that

$$\frac{V}{U}\le\frac{m}{n-pt}\le\frac{d}{1-(1-d)/2}=\frac{1}{1/2+1/(2d)}<1. \tag{9}$$

Also

$$u_i^s=\Theta(k)=\Theta(n/p)=\Theta\left(\frac{2nt}{m(1-d)}\right)=\omega(1).$$

To estimate v_i^s, we use Corollary 4.1

$$\left|v_i^s-u_i^s\frac{V}{U}\right|\le\frac{2u_i^s}{k}\le 4$$

so that

$$v_i^s\ge\frac{kV}{U}-4\ge\frac{n}{p}\frac{m-pt}{n}-4=\frac{m}{p}-t-4\ge t-4=\omega(1).$$

Also note that since $k=\omega(1)$ and V/U is bounded away from 1 by a constant by (9), it follows from Corollary 4.1 that $v_i^s/u_i^s<c$ for some constant $c<1$ for all i. Next, we can employ the Stirling approximation

$$x!=\Theta\left(\sqrt{x}(x/e)^x\right)$$

for the binomial coefficients from (8) as follows:

$$\binom{u}{v}=\Theta\left(\frac{1}{\sqrt{v}\sqrt{1-v/u}}\frac{(u/e)^u}{(v/e)^v((u-v)/e)^{u-v}}\right)=\Theta\left(\frac{1}{\sqrt{v}}\left(\frac{1}{\xi}\right)^v\left(\frac{1}{1-\xi}\right)^{u-v}\right),$$

where $\xi=v/u$. Now we can bound

$$\frac{\binom{u_1^s}{v_1^s}\binom{u_2^s}{v_2^s}\cdots\binom{u_q^s}{v_q^s}}{\binom{U}{V}}=\Theta(1)^q\frac{\sqrt{V}}{\prod_i\sqrt{v_i^s}}\frac{\phi^V(1-\phi)^{U-V}}{\prod_i{\xi_i}^{v_i^s}(1-\xi_i)^{u_i^s-v_i^s}}, \tag{10}$$

where $\phi=V/U<c$ and $\xi_i=v_i^s/u_i^s$. We estimate the product in the denominator using an inner point $\psi\in[\min_i\{\xi_i\},\max_i\{\xi_i\}]$ as follows:

$$\prod_i{\xi_i}^{v_i^s}(1-\xi_i)^{u_i^s-v_i^s}=\psi^V(1-\psi)^{U-V}. \tag{11}$$

This inner point exists, since all the functions in the corresponding expression are continuous. By Corollary 4.1, $|\psi-\phi|<2/k$. Now, we bound the expression

$$\begin{aligned}\left(\frac{\phi}{\psi}\right)^V\left(\frac{1-\phi}{1-\psi}\right)^{U-V}&=\left(1+\frac{\phi-\psi}{\psi}\right)^V\left(1+\frac{\psi-\phi}{1-\psi}\right)^{U-V}\\&=\left(1+O\left(\frac{1}{t}\right)\right)^V\left(1+O\left(\frac{1}{k}\right)\right)^U\\&=\exp\left(O\left(\frac{m}{t}+\frac{n}{k}\right)\right)=2^{O(q)}\end{aligned}$$

since $\psi\ge\min_i\{v_i^s/u_i^s\}\ge(t-4)/2k=\Omega(t/k)$ and $1-\psi>1-c$.

Now, we can bound (10) by

$$\frac{\binom{u_1^s}{v_1^s}\binom{u_2^s}{v_2^s}\cdots\binom{u_q^s}{v_q^s}}{\binom{U}{V}} = \frac{\sqrt{m}}{\Omega(t)^{q/2}}.$$

For a given group of leaves G_y, we bound the total number of compatible bit vectors by (10). So that the total number of compatible bit vectors for all groups is at most

$$2^r \sum_{y=0}^{pt} \binom{pt}{y}\binom{n-pt}{m-y}\frac{\sqrt{m}}{\Omega(t)^{q/2}} = 2^r\binom{n}{m}\frac{\sqrt{m}}{\Omega(t)^{q/2}}.$$

However, all possible bit vectors of length n with m ones have to be compatible with at least one leaf, so that

$$r = \frac{p}{8}\lg t - \frac{\lg m}{2} + \Omega(m/t) = \frac{m(1-d)}{16t}\lg t - \frac{\lg m}{2} + \Omega(m/t).$$

4.1. Select index

To show a lower bound on the select index, we simulate p queries select(ik) for $i = 1, \ldots, p$, where $k = m/p$. We define *blocks* similar to Section 3.2, namely the ith block is from position select$_B((i-1)k)+1$ to position select$_B(ik)$, so that the *cardinality* of each block (the number of 1-bits in it) is exactly k. Similar to the case of rank, we define *superblocks*. The ith superblock (except, perhaps, the last one that could be of cardinality smaller than k) will contain consecutive blocks $z_{i-1}+1, \ldots, z_i$, such that $k \le v_{z_{i-1}+1} + v_{z_{i-1}+2} + \cdots u_{z_i} \le 2k$ and

$$v_i^s = v_{z_{i-1}+1} + v_{z_{i-1}+2} + \cdots + v_{z_i}.$$

Note that this is always possible, since $v_i \le k$ for all i. And hence q, the number of superblocks, is at least $V/(2k)$. The *size* of ith superblock is given by

$$u_i^s = u_{z_{i-1}+1} + u_{z_{i-1}+2} + \cdots + u_{z_i}.$$

We use inequality (8) to bound the number of bit vectors compatible with a given leaf. Now we only need to bound the right side of (8)

$$\binom{u_1^s}{v_1^s}\binom{u_2^s}{v_2^s}\cdots\binom{u_q^s}{v_q^s}. \tag{12}$$

Similar to the case of the rank index, we maximize this expression for fixed values of v_i^s's subject to the constraint

$$u_1^s + u_2^s + \cdots + u_q^s = U.$$

In order to do so, we show a counterpart of Lemma 4.1 and Corollary 4.1.

Lemma 4.2. *At a local maximum of* (12)*, we have*

$$\frac{u_j^s + 1}{v_j^s} \ge \frac{u_i^s}{v_i^s}$$

for each $i \ne j$.

Proof.

$$\binom{u_i^s}{v_i^s}\binom{u_j^s}{v_j^s} \ge \binom{u_i^s - 1}{v_i^s}\binom{u_j^s + 1}{v_j^s} \quad \text{simplifies to}$$

$$\frac{u_i^s}{u_i^s - v_i^s} \ge \frac{u_j^s + 1}{u_j^s - v_j + 1} \quad \text{and the result follows.} \quad \square$$

Corollary 4.2. *At a local maximum of* (12)*, we have*

$$\left|\frac{v_i^s}{u_i^s} - \frac{V}{U}\right| = O\left(\frac{V^2}{kU^2}\right).$$

Proof. First note that for each $i \neq j$, we have

$$\frac{u_j^s}{v_j^s} + \frac{1}{v_j^s} \geq \frac{u_i^s}{v_i^s} \geq \frac{u_j^s}{v_j^s} - \frac{1}{v_i^s}.$$

Since v_i^s and v_j^s are at least k, we have

$$\left|\frac{u_i^s}{v_i^s} - \frac{u_j^s}{v_j^s}\right| \leq \frac{1}{k} \quad \text{and} \quad \text{hence} \quad \left|\frac{u_i^s}{v_i^s} - \frac{U}{V}\right| \leq \frac{1}{k}.$$

Since $1/k = o(1)$, we have

$$\left|\frac{v_i^s}{u_i^s} - \frac{V}{U}\right| = \left|\frac{u_i^s}{v_i^s} - \frac{U}{V}\right| \frac{v_i^s V}{u_i^s U}$$

and the result follows. □

Similar to the case of the rank index, we choose $p = m(1-d)/2t$, and $k = m/p = 2t/(1-d) = \omega(1)$. Using an internal point ψ similar to (11) we get

$$\frac{\binom{u_1^s}{v_1^s}\binom{u_2^s}{v_2^s}\cdots\binom{u_q^s}{v_q^s}}{\binom{U}{V}} = \Theta(1)^q \frac{\sqrt{m}}{t^{q/2}} \left(\frac{\phi}{\psi}\right)^V \left(\frac{1-\phi}{1-\psi}\right)^{U-V}, \tag{13}$$

where $\phi = V/U$ and ψ is a number, such that $|\psi - \phi| = O\left(\frac{V^2}{kU^2}\right)$ by Corollary 4.2. From Eq. (9), we conclude that both ϕ and ψ are bounded away from 1 by a constant $c < 1$. We also have $\psi = \Omega(V/U)$ by Corollary 4.2.

$$\begin{aligned}\left(\frac{\phi}{\psi}\right)^V \left(\frac{1-\phi}{1-\psi}\right)^{U-V} &= \left(1 + \frac{\phi-\psi}{\psi}\right)^V \left(1 + \frac{\psi-\phi}{1-\psi}\right)^{U-V} \\ &= \left(1 + O\left(\frac{V}{kU}\right)\right)^V \left(1 + O\left(\frac{V^2}{kU^2}\right)\right)^U \\ &= \exp\left(O\left(\frac{V^2}{kU}\right)\right).\end{aligned}$$

Note that $m \geq V \geq m - pt > m/2$ and $U = n - pt = \Theta(n)$, thus $V^2/(kU) = m/k \cdot m/n = O(q)$ Therefore, the left side of (13) is at most $\frac{\sqrt{m}}{\Omega(t)^{q/2}}$, thus we have

$$r = \frac{p}{8}\lg t - \frac{\lg m}{2} + \Omega(m/t) = \frac{m(1-d)}{16t}\lg t - \frac{\lg m}{2} + \Omega(m/t)$$

since $q \geq V/(2k) \geq m/(4k) = p/4$.

We conclude with

Theorem 4.1. *Let B be a bit vector of length n with $m \leq dn$ ones in it for a constant $d < 1$. Assume that there is an algorithm that uses $t = \omega(1)$ bit probes to B plus unlimited access to an index of size r and unlimited computation power to answer rank (respectively, select) queries. Then $r = \frac{m(1-d)}{16t}\lg t - \frac{\lg m}{2} + \Omega(m/t)$.*

Note that this theorem gives an optimal lower bound for the case of constant density bit vectors (i.e. when $m/n < c$, for constant $c < 1$). For the select index, it also yields a lower bound better than the one given by Miltersen [3] for the case where $m = \Omega(nt/(\lg n \lg t))$, or $m = \Omega(n/\lg\lg n)$ when $t = \Theta(\lg n)$.

Acknowledgments

We thank Ian Munro, Prabhakar Ragde, and Jérémy Barbay for fruitful discussions and proof reading of this paper. We especially thank Jérémy Barbay for proof reading the final journal version and pointing out several important typos. We thank S. Srinivasa Rao for bringing the problem to our attention and pointing out the results of Miltersen. We thank Peter Widmayer for helpful discussions while the author was visiting ETH in Zürich, in particular for suggestions on how to improve the presentation of the indexing model and for pointing out potential applications to shortest path queries on large graphs with a "road map" playing the role of an index. We also thank the anonymous referees for their helpful comments.

Uncited figures

Figs. 1 and 2

References

[1] David R. Clark, Compact pat trees, Ph.D. Thesis, University of Waterloo, 1996.
[2] Guy Jacobson, Succinct static data structures, Ph.D. Thesis, Carnegie Mellon University, January 1989.
[3] Peter Bro Miltersen, Lower bounds on the size of selection and rank indexes, in: Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms, 2005, pp. 11–12.
[4] J. Ian Munro, Venkatesh Raman, S. Srinivasa Rao, Space efficient suffix trees, in: Proceedings of the 18th Conference on the Foundations of Software Technology and Theoretical Computer Science, in: Lecture Notes in Computer Science, vol. 1530, Springer, 1998, pp. 186–196.
[5] J. Ian Munro, Venkatesh Raman, S. Srinivasa Rao, Space efficient suffix trees, Journal of Algorithms 39 (2) (2001) 205–222.
[6] Rajeev Raman, Venkatesh Raman, S. Srinivasa Rao, Succinct indexable dictionaries with applications to encoding k-ary trees and multisets, in: Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms, 2002, pp. 233–242.
[7] Peter Widmayer, Personal communication, 2006.